

Lucas Santana Leal

Quicksort

EM ASSEMBLY

O que é Quicksort

Quicksort é um algoritmo de ordenação que adota a estratégia de divisão e conquista que possui pior caso $O(n^2)$ e melhor caso $O(n\log n)$, operando da seguinte forma:

1. Escolhe um pivô, um elemento do array, a partir de uma dessas opções:
 - Primeiro elemento do array. (menos eficiente)
 - Usando a fórmula $(\text{low} + \text{high})/2$, onde low e high são os valores do primeiro e do último elemento do array respectivamente. (pega o elemento do meio, sendo mais eficiente)
 - Usando a fórmula $\text{low} + (\text{high} - \text{low})/2$, onde low e high são os valores do primeiro e do último elemento do array respectivamente. (também pega o elemento do meio, mas evita overflow se os valores forem muito altos)
2. Particiona o array, rearranjando a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores que ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sub listas não ordenadas.
3. Faz duas chamadas recursivas, em uma ordena a sub lista dos elementos menores e na outra ordena a sub lista dos elementos maiores

Métodos Usados

O Quicksort feito utiliza um array dinâmico (tamanho definido na memória e elementos definidos durante a execução, recebendo das entradas do teclado), o pivô é o primeiro elemento do array e usa o particionamento de Hoare que utiliza dois ponteiros que convergem (andam em direção um ao outro) vindo das extremidades do array.

Visto isso, pode-se dizer que esse Quicksort não é o mais eficiente possível, pois poderia ter uma escolha melhor de pivô e usar outras técnicas de particionamento, como a 3-Way e Dual-Pivot. Porém, opera de forma média, é fácil de se compreender e prático para codificar.

O código base em C usado para ser traduzido em assembly MIPS se encontra aqui: [código-fonte](#)

Como foi feito o código em assembly

Como o código base escolhido é em C, houve extrema facilidade de traduzir para o assembly MIPS. Existem algumas formas de conseguir executar um código em assembly, como através de linhas de comando no terminal ou usando uma IDE própria. Para esse projeto, como o assembly escolhido foi o MIPS, foi usada a IDE MARS (MIPS Assembler and Runtime Simulator) que pode ser baixada em [MARS Download](#).

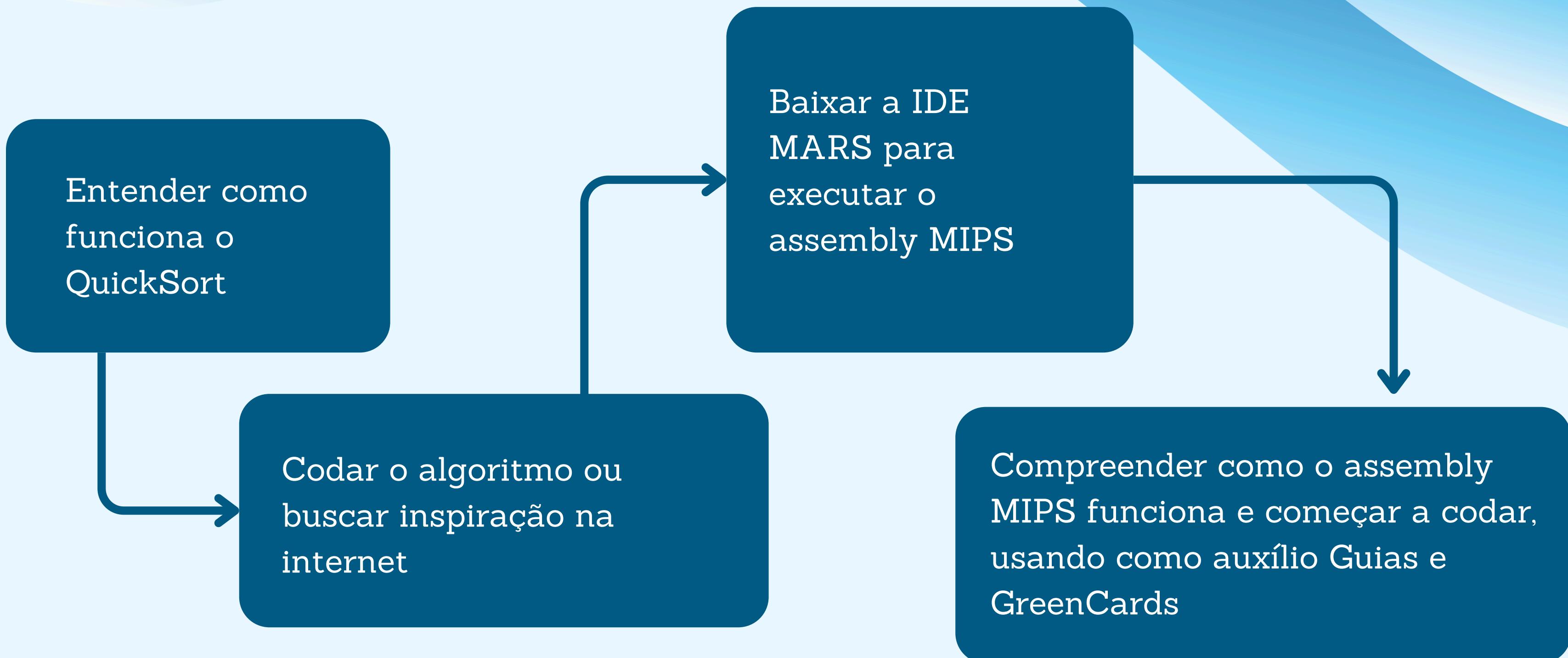
A IDE opera em Java e a versão usada nesse projeto foi:

openjdk version "25.0.1" 2025-10-21

OpenJDK Runtime Environment (Red_Hat-25.0.1.0.8-3) (build 25.0.1+8)

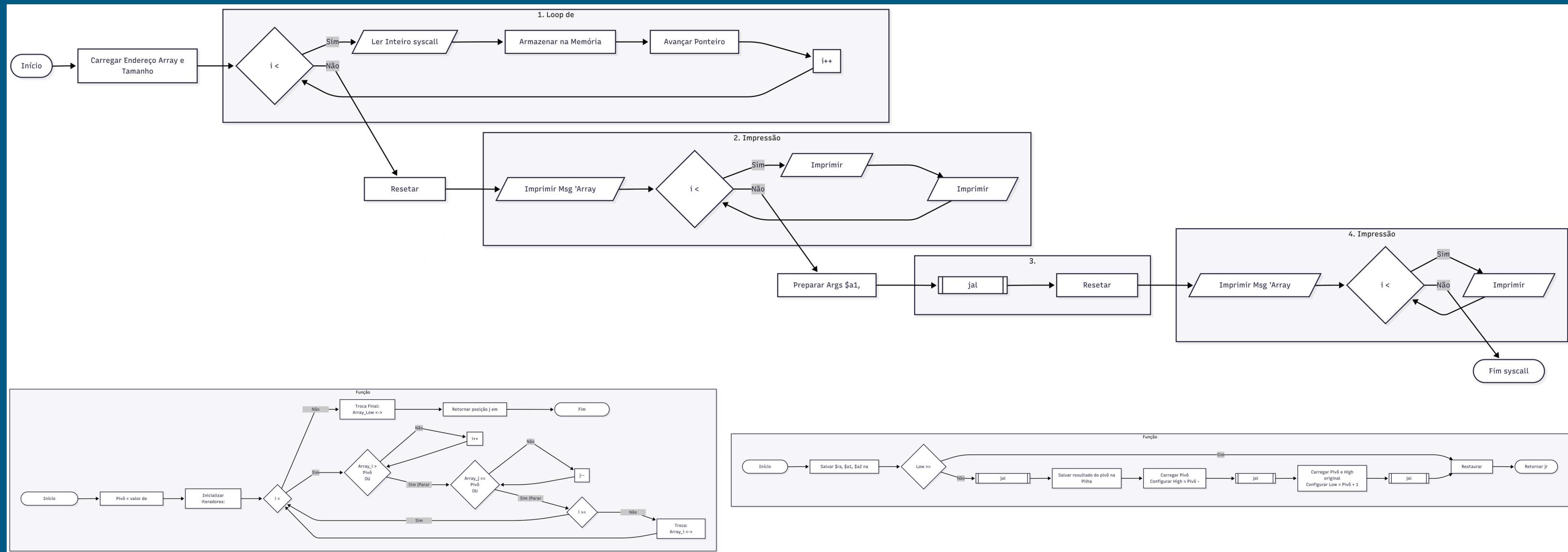
OpenJDK 64-Bit Server VM (Red_Hat-25.0.1.0.8-3) (build 25.0.1+8, mixed mode, sharing)

Fluxo até então



O assembly pode ser visto e explicado em: [código](#)

Diagrama de blocos a seguir para guiar-se:



Pontos Importantes

1. O código usa recursão, então o maior desafio enfrentado foi compreender como a manipulação da pilha acontece (abertura e fechamento dos argumentos e trabalho do stack pointer \$sp).
2. Como dito anteriormente, esse código não é o mais eficiente possível, porém, uma forma de melhorá-lo seria modificar a escolha do pivô após digitar todos os elementos do array.
3. O assembly não está o mais limpo e otimizado possível, visto que as vezes há uso excessivo ou desnecessário de registradores.